# Smart Contract Security Audit Report

# Table Of Contents

**1 Executive Summary**

**2 Audit Methodology**

**3 Project Overview**

3.1 Project Introduction

3.2 Vulnerability Information

**4 Code Overview**

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

**5 Audit Result**

**6 Statement**

# 1 Executive Summary

On 2024.01.30, the SlowMist security team received the 9GAG team's security audit application for Memecoin Staking, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

| Test method | Description |
|---|---|
| Black box testing | Conduct security tests from an attacker's perspective externally. |
| Grey box testing | Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses. |
| White box testing | Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc. |

The vulnerability severity level information:

| Level | Description |
|---|---|
| Critical | Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities. |
| High | High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities. |
| Medium | Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities. |
| Low | Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed. |
| Weakness | There are safety risks theoretically, but it is extremely difficult to reproduce in engineering. |
| Suggestion | There are better practices for coding or architecture. |

# 2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

| Serial Number | Audit Class | Audit Subclass |
|---|---|---|
| 1 | Overflow Audit | - |
| 2 | Reentrancy Attack Audit | - |
| 3 | Replay Attack Audit | - |
| 4 | Flashloan Attack Audit | - |
| 5 | Race Conditions Audit | Reordering Attack Audit |
| 6 | Permission Vulnerability Audit | Access Control Audit |
| | | Excessive Authority Audit |
| 7 | Security Design Audit | External Module Safe Use Audit |
| | | Compiler Version Security Audit |
| | | Hard-coded Address Security Audit |
| | | Fallback Function Safe Use Audit |
| | | Show Coding Security Audit |
| | | Function Return Value Security Audit |
| | | External Call Function Security Audit |

| Serial Number | Audit Class | Audit Subclass |
|---|---|---|
| 7 | Security Design Audit | Block data Dependence Security Audit |
| | | tx.origin Authentication Security Audit |
| 8 | Denial of Service Audit | - |
| 9 | Gas Optimization Audit | - |
| 10 | Design Logic Audit | - |
| 11 | Variable Coverage Vulnerability Audit | - |
| 12 | "False Top-up" Vulnerability Audit | - |
| 13 | Scoping and Declarations Audit | - |
| 14 | Malicious Event Log Audit | - |
| 15 | Arithmetic Accuracy Deviation Audit | - |
| 16 | Uninitialized Storage Pointer Audit | - |

# 3 Project Overview

## 3.1 Project Introduction

This is the staking protocol of Memecoin, including Claim, Delegation and Staking parts.

## 3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

| NO | Title | Category | Level | Status |
|---|---|---|---|---|
| N1 | Risk of excessive authority | Authority Control Vulnerability Audit | Medium | Acknowledged |
| N2 | Return value not checked | Others | Suggestion | Fixed |

| NO | Title | Category | Level | Status |
|---|---|---|---|---|
| N3 | Redundant code | Others | Suggestion | Fixed |
| N4 | No zero address check | Others | Suggestion | Fixed |

# 4 Code Overview

## 4.1 Contracts Description

https://github.com/9gag/memecoin-staking-audit

Initial audit commit: 24e20eca249c33d89d04ef9149117bcb8d22ad9e

FInal audit commit: 62237b4a86fd95dea1bc68e3024a750145d02fd9

The main network address of the contract is as follows:

**The code was not deployed to the mainnet.**

## 4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

| **MemecoinMultiClaim** | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | MemecoinDelegatable |
| multiClaim | External | Can Modify State | - |
| _getRequester | Private | - | - |
| multiClaimToStakeland | External | Can Modify State | - |

| **MemecoinDelegatable** | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |

**MemecoinDelegatable**

| | | | |
|---|---|---|---|
| <Constructor> | Public | Can Modify State | - |
| delegate | External | - | - |
| _delegateTransfer | Internal | Can Modify State | - |
| _delegatePermit | Internal | Can Modify State | - |

**MemecoinDelegatableUpgradeable**

| Function Name | Visibility | Mutability | Modifiers |
|---|---|---|---|
| __MemecoinDelegatable_init | Internal | Can Modify State | onlyInitializing |
| __MemecoinDelegatable_init_unchained | Internal | Can Modify State | onlyInitializing |
| delegate | External | - | - |
| _delegateTransfer | Internal | Can Modify State | - |
| _delegatePermit | Internal | Can Modify State | - |

**MemecoinDelegate**

| Function Name | Visibility | Mutability | Modifiers |
|---|---|---|---|
| <Constructor> | Public | Can Modify State | - |
| transferFrom | External | Can Modify State | onlyAuthorized |
| allowance | External | - | - |
| isAuthorized | External | - | - |

**MemecoinStaking**

| Function Name | Visibility | Mutability | Modifiers |
|---|---|---|---|
| _authorizeUpgrade | Internal | Can Modify State | onlyUpgrader |

| MemecoinStaking | | | |
|---|---|---|---|
| <Constructor> | Public | Can Modify State | - |
| initialize | External | Can Modify State | initializer |
| stake | External | Can Modify State | nonReentrant onlyValidStakingSetup onlyValidAmount |
| stakeFor | External | Can Modify State | nonReentrant onlyValidStakingSetup onlyValidAmount onlyDelegatable |
| _stake | Private | Can Modify State | - |
| unstake | External | Can Modify State | nonReentrant onlyValidStakingSetup onlyValidAmount |
| _unstake | Private | Can Modify State | - |
| _redeemRewards | Private | Can Modify State | - |
| _verifyProof | Private | - | - |
| stakeRewards | External | Can Modify State | onlyOwner |
| setStakingActive | External | Can Modify State | onlyOwner |
| setStakingStartDate | External | Can Modify State | onlyOwner |
| setUpgrader | External | Can Modify State | onlyOwner |
| renounceUpgrader | External | Can Modify State | onlyOwner |
| totalSupply | External | - | - |
| stakeOf | External | - | - |
| getRewardRedeemedAt | External | - | - |

## 4.3 Vulnerability Summary

### [N1] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability Audit**

**Content**

In the MemecoinStaking contract, the Owner role can modify important parameters in the contract.

- MemecoinStaking.sol#L204-L208,L211-L216,L222-L228,L233-L240

```
function setStakingActive
function setStakingStartDate
function setUpgrader
function renounceUpgrader
```

Since the MemecoinStaking contract adopts the UUPS upgrade mode, the upgrader role can upgrade the contract.

- MemecoinStaking.sol#L47

```
function _authorizeUpgrade(address) internal override onlyUpgrader {}
```

**Solution**

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

**Status**

Acknowledged

### [N2] [Suggestion] Return value not checked

**Category: Others**

**Content**

In the MemecoinMultiClaim contract, the `constructor` function does not check the return value when calling the `approve` function of the memecoin token contract.The `_redeemRewards` function and the `stakeOf` function don't check the return value when calling the `_verifyProof` function.

- MemecoinMultiClaim.sol#L39-L49,L130-L147,L252-L266

```
    constructor(address _presaleClaim, address _airdropClaim, address _memecoin,
address _delegate, address _staking)
        MemecoinDelegatable(_delegate)
    {
        presaleClaim = IMemecoinClaim(_presaleClaim);
        airdropClaim = IMemecoinClaim(_airdropClaim);
        dc = IDelegationRegistry(0x00000000000076A84feF008CDAbe6409d2FE638B);
        dcV2 = IDelegateRegistry(0x00000000000000447e69651d841bD8D104Bed493);
        memecoin = IERC20(_memecoin);
        memecoin.approve(_delegate, type(uint256).max);
        staking = IMemecoinStaking(_staking);
    }

    function _redeemRewards(address user, Reward[] calldata rewards) private {
        for (uint256 i; i < rewards.length; i++) {
            Reward calldata reward = rewards[i];
            uint256 rewardId = reward.rewardId;

            if (usersRewardRedeemedAt[user][rewardId] > 0) continue;

            uint256 amount = reward.amount;
            _verifyProof(user, rewardId, amount, reward.proof);

            unchecked {
                balanceOf[user] += amount;
            }
            emit Transfer(address(this), user, amount);
            usersRewardRedeemedAt[user][rewardId] = block.timestamp;
            emit RewardRedeemed(user, rewardId, amount, block.timestamp);
        }
    }

       function stakeOf(address user, Reward[] calldata rewards) external view
returns (uint256 balance) {
        balance = balanceOf[user];
        if (rewards.length != 0) {
            for (uint256 i; i < rewards.length; i++) {
                Reward calldata reward = rewards[i];
                uint256 amount = reward.amount;
                uint256 rewardId = reward.rewardId;

                if (usersRewardRedeemedAt[user][rewardId] > 0) continue;
                _verifyProof(user, rewardId, amount, reward.proof);

                balance += amount;
            }
        }
    }
```

**Solution**

It is recommended to check the return value.

**Status**

Fixed

### [N3] [Suggestion] Redundant code

**Category: Others**

**Content**

In the MemecoinStaking contract, the three variables `name`, `symbol`, and `decimals` are not used.

- MemecoinStaking.sol#L42-L44

```
    string public constant name = "Staked Memecoin";
    string public constant symbol = "";
    uint8 public constant decimals = 18;
```

**Solution**

It is recommended to remove redundant code.

**Status**

Fixed; These code have no clear context in the audited commit, but clear context is provided in the final version.

### [N4] [Suggestion] No zero address check

**Category: Others**

**Content**

In the MemecoinStaking contract, the `initialize` function doesn't perform a zero address check on the `_delegate` parameter.

- MemecoinStaking.sol#L54-L62

```
function initialize(address _memecoin, address _delegate) external initializer {
        ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained();
        OwnableUpgradeable.__Ownable_init_unchained();
        UUPSUpgradeable.__UUPSUpgradeable_init();
        MemecoinDelegatableUpgradeable.__MemecoinDelegatable_init(_delegate);

        if (_memecoin == address(0)) revert InvalidAddress();
        memecoin = IERC20(_memecoin);
    }
```

**Solution**

It is recommended to perform a zero address check on the _delegate parameter

**Status**

Fixed

## 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
|---|---|---|---|
| 0X002402020001 | SlowMist Security Team | 2024.01.30 - 2024.02.02 | Medium Risk |

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 3 suggestion vulnerabilities.

# 6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

**Official Website**
www.slowmist.com

**E-mail**
team@slowmist.com

**Twitter**
@SlowMist_Team

**Github**
https://github.com/slowmist